Foundations of Programming Languages – Course Overview

Acknowledgments: some slides taken or adapted from lecture notes of Stanford CS242
https://courseware.stanford.edu/pg/courses/317431/

What are programming languages for

- Communication between programmers and HW
  - Model the real world
  - Model computation & communication
- One of the most fundamental area of computer science
- Examples
  - assembly, imperative (e.g., C), functional, OO, logical, web (e.g., JavaScript), domain-specific languages
- Still a very active field, both in academia and industry
  - New languages: F#, Go, Scala, …

POPL 2017 Invited Talk on Rust

POPL 2017 (series) / POPL 2017 /

Rust: from POPL to practice

Track POPL 2017

When **Fri 20 Jan 2017 09:05 - 10:00 at Auditorium** - Invited speaker Chair(s): Giuseppe Castagna

Abstract

In 2015, a language based fundamentally on substructural typing–Rust–hit its 1.0 release, and less than a year later it has been put into production use in a number of tech companies, including some household names. The language has started a trend, with several other mainstream languages, including C++ and Swift, in the early stages of incorporating ideas about ownership. How did this come about?

Rust's core focus is safe systems programming. It does not require a runtime system or garbage collector, but guarantees memory safety. It does not stipulate any particular style of concurrent programming, but instead provides the tools needed to guarantee data race freedom even when doing low-level shared-state concurrency. It allows you to build up high-level abstractions without paying a tax; its compilation model ensures that the abstractions boil away.

These benefits derive from two core aspects of Rust: its ownership system (based on substructural typing) and its trait system (a descendant of Haskell's typeclasses). The talk will cover these two pillars of Rust design, with particular attention to the key innovations that make the language usable at scale. It will highlight the implications for concurrency, where Rust provides a unique perspective. It will also touch on aspects of Rust's development that tend to get less attention within the POPL community: Rust's governance and open development process, and design considerations around language and library evolution. Finally, it will mention a few of the myriad open research questions around Rust.

Aaron Turon

MPI-SWS

Session Program

Fri 20 Jan

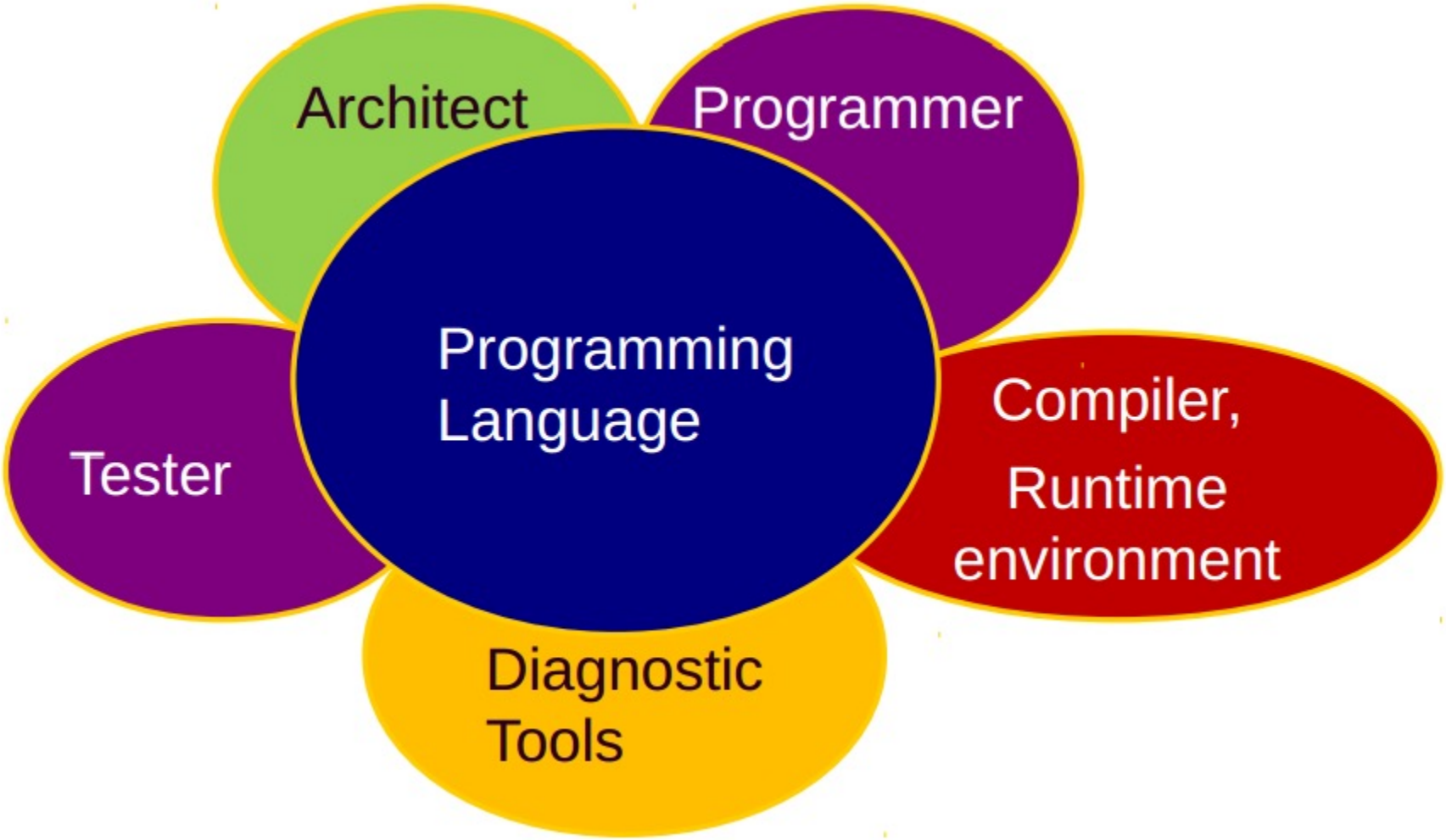
09:05 - 10:00: POPL - Invited speaker at Auditorium
Chair(s): Giuseppe Castagna

09:05 - 10:00 *Talk* ☆ **Rust: from POPL to practice**
Aaron Turon

What do we care

- Easy to use
  - Language design:
    good syntax, clear semantics, high-abstraction level
  - Enhance software productivity
    - e.g., domain specific languages (DSL)
- Better performance
  - Language implementations:
    compilers, runtime (GC), parallelization
- Better software quality (reliability and security)
  - Type safety, static/dynamic checking, verification
- Theoretical foundations
  - Semantics, verification, etc.
    - Connections with other related fields: logic, computation theory, etc.

Language goals and trade-offs



Why should you take this course

- Programming language concepts
  - A language is a “conceptual universe” (Perlis)
    - OO vs. Functional, for instance
  - Distinguish key properties from superficial details
- Better programming skills
  - Write more efficient and reliable code
  - Be prepared for new PL methods, paradigms, tools
- Learn to design your own languages
  - Domain-specific languages (e.g., big data, machine learning, networking, robotics)

Some PL Research Goals

- Design and Implementation
  - Easy to use (design), efficient executable code (impl)
  - Multicore/Parallel/Distributed programming
  - Flaw detection: static, dynamic, etc.
  - Related fields: OS, architecture, domain specific fields
- Principles and Theories
  - Semantics and Properties (e.g. expressiveness) of Programming Languages
  - Principles and theories for safety/security/correctness
  - Program analysis and verification
  - Related fields: logic and algebra, computation theory

Major Conferences

- Principles of Programming Languages (POPL)
- Programming Language Design and Implementation (PLDI)
- Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)
- Principles and Practice of Parallel Programming (PPoPP)
- International Conferences on Functional Programming (ICFP)
- Architectural Support for Programming Languages and Operating Systems (ASPLOS)
- Languages, Compilers and Tools for Embedded Systems (LCTES)

Major Conferences (2)

- Related:
  - Logic in Computer Science (LICS)
  - Computer Aided Verification (CAV)

Temporary Syllabus

- Introduction
- Haskell
- Foundations: lambda calculus, opr. semantics
- Scope and stack storage allocation
- Types and type checking/inference
- Parametric polymorphism, type classes (ad-hoc polymorphism)
- Monads
- Exceptions and continuations

Temporary Syllabus (2)

- Modularity
- Objects
- Prototypes, classes, inheritance
- Object types and subtyping
- Implementation structures
- Templates and generics
- Concurrency & Atomicity
- Advanced topics